**K8S BASIC COMPONENTS**

* Pod is an abstract on a container in a node.
* Each component has its own pod. Each Pod has its own IP address, but the limitation is that once a pod die (because they are ephemeral, you will need always adjust the IP to match the new one assigned once restarted). As a result, “service” is assigned to each pod.
* A service a way to set a static/permanent IP address assigned to each pod. The life cycle of the service and the pod are not attached. In a distributed and multiple instances of a Node, the service also serves as a load balancer – forward traffic to less busy instances or appropriate instances.
* External service (external access) and internal service (internal access)
* Ingress can also be used for forwarding traffic.

**ConfigMap and Secret**

* ConfigMap : external config to the application. Contains URL of DB and other services. You can add this to the pod (of the app).
* Secret : like config map used to store credentials in base64 encoded. You need to connect it to your pod as well.

**Data storage**

If DB pod is restarted, data is gone. For persistent data, use VOLUMES. Volumes attaches a physical storage to your POD on a hard drive on a local machine/cloud – which is not part of the pod itself.

**Deployments**

This is an abstraction of PODS. Deployment is what you create, where you can create pods, specify numbers that you want to create and scale up and down. DB cannot be replicated with a deployment because DB has a state. StatefulSet is used for APPs like DBs. DBs should be created using StatefulSet and not deployment.

**K8S ARCHITECTURE**

Nodes have 2 types of Nodes (Master and Slave).

Each nodes have multiple pods running on its. Nodes can be called worker nodes and they require processes to be installed for it to work. These processes are :

* Container runtime e.g., Docker – needs to be installed on every pod
* Kubelet interacts with both container and the node and assigns resources to the pod. It runs the pod using the container runtime--- MUST be installed on every Kubernetes worker node
* Kube Proxy forwards request from services to the pod -- MUST be installed on every Kubernetes worker node

**Master Nodes** manages processes and have 4 process that manages process on the worker nodes and cluster state.

* **API server** – like cluster gateway, acts a gatekeeper as authentication, validates request before forwarding to required services/other processes. You can interact with the API server via UI, API or KUBECTL.
* **Scheduler** – receives requests from API server. Intelligent enough to know what worker node the new pod will be scheduled. It just decides where the New POD will be put
* **Controller manager** – used to detect state changes like crashing/death and tries to recover them as soon as possible.
* **Etcd** – a key value store for a cluster state. Every change in a cluster is updated and saved in the etcd. It is like the cluster brain. NOTE: Actual application data is not stored here

Graphical user interface, diagram

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

**MINIKUBE and KUBECTL**

Minikube : an OSS tool. 1 node K8S cluster in which both the master and the worker process run. It has docker runtime pre-installed. It runs through a virtual box (creates it in your laptop). Great for testing.

Kubectl : To create pod and other k8s components, use kubectl – a command line tool. You can interact with the API server using Kubectl. It can also be used for cloud or hybrid cluster.

Commands :

Minikube : lists the minikube commands and how they can be used

Minikube start –vm-driver=hyperv : instructs the minikube to start the cluster using the hyperv vm driver

Note : minikube is just used to start and delete the cluster, other configuring will be using kubectl.

Kubectl : list the kubectl commands and how they can be used

**Basic kubectl commands**

Kubectl get nodes

Kubectl get pod

Kubectl get services

Kubectl get deployments

Kubectl create deployment <name of deployment> –image=image (e.g., kubectl create deployment nginx-deploy –image=nginx

Note: deployment has the blueprint to create the pod

Kubectl get replicaset (manages the replica of the pod)

Kubectl edit deployment <name> -- a notepad or an editor pops up, once you edit the deployment configuration and close the notepad or editor, it automatically acknowledges the edit and execute the new deployment.

Kubectl logs <name of deployment> -- 2nd image

Kubectl describe pod <name of deployment> (get a description of the deployment) –1st image

Text

Description automatically generated

Text

Description automatically generated

Kubectl exec -it <name of deployment> --bin/bash (it means interactive) – This takes take you to the cli of the deployment (deployed app). This is also useful in debugging.

Kubectl delete deployment <name of deployment> ---- to delete a deployment, its pods and all replicasets

In practice, you will need to work with a configuration file to specify the names and other options of the deployment you want to create. In this case, you will use the command below:

Kubectl apply -f config-file.yaml